Cucumber BDD Framework building documentation:

Step 1: Creating Maven Project

1. Create a Maven project in eclipse
2. Select default template as 'maven-archetype-quickstart'
3. GroupID- Organizantion
4. ArtifactID- Application
5. Package- anyName

Step 2: Create an sample Selenium test involving multiple pages and keep it as reference

Step 3: Updating POM.XML with necessary dependencies from mvnRepository

1. Selenium Java (For selenium libraries)
2. io.github.bonigarcia- Webdriver manager(To get latest drivers at runtime)
3. Cucumber Java (For incorporating cucumber BDD)
4. Cumber TestNG (Runner needs either TestNG or Junit)
5. Apache commons.io (Utilities for file manipulations)
6. Cucumber pico container- (Dependency Injection and Factory design pattern)
7. Aventastck Extent reports (Reporting framework)
8. Tech.grasshopper cucumber 7 adapter (Extent report adapter)

Step 4: Create the Feature Files

1. Create a new package under test/java for feature files(cucumber.features)
2. Create a folder(optional) to have the feature files in the folder as per the type of test
    a. Eg: StandaloneTests, EndToEndTests, etc
3. Create necessary feature files – 1 for each Module/userstory

```
Feature: Title of your feature
  I want to use this template for my feature file
  @tag1
  Scenario: Title of your scenario
    Given I want to write a step with precondition
    And some other precondition
    When I complete action
    And some other action
    And yet another action
    Then I validate the outcomes
    And check more outcomes

  @tag2
  Scenario Outline: Title of your scenario outline
    Given I want to write a step with <name>
    When I check for the <value> in step
    Then I verify the <status> in step

    Examples:
      | name  | value | status  |
      | name1 |     5 | success |
       | name2 |     7 | Fail    |
```

Step 5: Runner File

1. Create a TestNG runner class in a new package(cucumber.runners) in test/java

Note: TestNG/Junit runner class should only be under test/java as Maven will only look under test/java to see the files that can be run using TestNG/Junit configuration

2. Extend the TestNG runner class with `AbstractTestNGCucumberTests`
3. Provide cucumberOptions annotation above the class declaration as below

```
@CucumberOptions(features = "src/test/java/cucumber/features",
glue="cucumber.stepDefenitions", tags= "@Regression", monochrome = true,
dryRun = false)
```

```
features-> path of the feature file from src
glue-> package name of the stepDefenitions
Note: Stepdefenition and features should be under test/Java
tags-> what tag need to be run
monochrome-> set to true to get cleaner console logs
dryRun-> set to true to get unimplemented step definition methods, set to
false during actual execution
```

4. Run the runnerFile with dryRun= True or run the feature file independently to get the missing implementations in the console

Step 6: Property reader and Global Parameters

1. Create a Global Parameters.properties property file in src/test/resources
2. Global Parameters property file will have information such as Browser, Environment, URL's for test, execution via grid etc.
3. Create a resources package in main/java (cucumber.resources)
4. Create a property reader class(PropertyReader.java) to read values from Global parameters and place under resources package

```
        public String readPropertyValue(String file, String key) throws
IOException
        {
                Properties prop= new Properties();
                FileInputStream fis = null;
                if (file.contains("global"))
                {
                fis= new FileInputStream(new
File(System.getProperty("user.dir")+"\\src\\test\\resources\\GlobalParamete
rs.properties"));
                }

                prop.load(fis);
                String value=prop.getProperty(key);
                return value;
        }
```

Step 7: Driver Manager

1. Create a Driver manager in main/java/cucumber.resources and inherit Property reader class
2. This class will have methods to invoke browser and to launch the application
   - Keep `WebDriver driver`; as global variable

- Create 1 method as private to invoke the browser as provided in Global Parameters and return the driver

```
    private WebDriver invokeBrowser() throws IOException {
            String browserProp = readPropertyValue("global", "Browser");
            String browserMaven= System.getProperty("browser");
            String bypass = readPropertyValue("global", "Bypass");
            int waitTimeValue = Integer.parseInt(readPropertyValue("global", 
"WaitTime"));
            Boolean flag = false;
            if (bypass.equalsIgnoreCase("true"))
                    flag = true;

            String browser= browserMaven!=null ? browserMaven:browserProp;


            if (browser.equalsIgnoreCase("chrome")) {
                    ChromeOptions options = new ChromeOptions();
                    options.setAcceptInsecureCerts(flag);
                    WebDriverManager.chromedriver().setup();
                    driver = new ChromeDriver(options);
            } else if (browser.equalsIgnoreCase("edge")) {
                    EdgeOptions options = new EdgeOptions();
                    options.setAcceptInsecureCerts(flag);
                    WebDriverManager.edgedriver().setup();
                    driver = new EdgeDriver(options);
            }

            driver.manage().window().maximize();
            driver.manage().deleteAllCookies();

    driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(waitTimeValue)
);

            return driver;

    }
```

- Create a LaunchApplication Method which will call the invokeBrowser method and assign it to driver
- From the environment value in the globalparameters.properties file, this method will navigate to the respective environment URL of the application

```
    public WebDriver launchApplication() throws IOException {
            String environment = readPropertyValue("global", "Environment");
            driver = invokeBrowser();
            if (environment.equalsIgnoreCase("PROD"))
                    driver.get(readPropertyValue("global", "PROD_URL"));
            else if (environment.equalsIgnoreCase("STAGE"))
                    driver.get(readPropertyValue("global", "STAGE_URL"));
            else if (environment.equalsIgnoreCase("QA"))
                    driver.get(readPropertyValue("global", "QA_URL"));
            else if (environment.equalsIgnoreCase("DEV"))
                    driver.get(readPropertyValue("global", "DEV_URL"));

            return driver;
    }
```

Step 8: Dependency Injection using io.cucumber.picocontainer

1. Create a new package called Globals in main/java in a new package (cucumber.globals)
2. Create a new class called Globals.java
3. Declare all the common variables including WebDriver driver in the class which can be accessed by all step definition files
    a. public WebDriver driver;
    b. public Boolean takeScreenshot;
    c. public int softErrors;

Step 9: Step Definition Files

1. Create a package under test/java (cucumber.stepDefenitions)
2. Create a java class called ApplicationGlobalSD-> this is the master stepDefenition class and where application global steps are defined like launching the application, Verifying if user is in a defined page, evaluating softAssertion errors etc.
3. Create a local variable of Globals class.

```
public Globals global;
```

4. Create a constructor for the ApplictionGlobalSD class and refer the Dependency file(Globals global)
5. Reference the instance variable to the parameter variable. This allows the driver object written in Globals.java to have the current state of the driver life.

```
public  ApplicationGlobalSD(Globals global) {

        this.global= global;

}
```

6. Application step definition will take object of DriverManager class and launch application
7. Create 1 stepdefenition file per page (later it will be referred to through the POM class files)
8. Create common stepdefenition for application common functions and create a generic step definition for generic functions used by all features

Note:

1. **Always copy ApplicationGlobalSD to create new stepDefenition files**
2. **All step definition files should have a constructor and have parameter of Globals class.**

Step 10: Base Actions and Utilities

1. Create a new package cucumber.commons in main/java
2. Create a Java class called Utilities with a local global variable WebDriver driver and a parameterized constructor with driver as the argument

```
public WebDriver driver;

    public Utilities(WebDriver driver)
    {
          this.driver=driver;
    }
```

3. All explicit wait statements and softAssert statements to be written in Utilities class
4. Base Action class will hold all the selenium wrapper methods required
    a. Will be inheriting utilities class

   b. Have a local global variable WebDriver driver and a parameterized constructor with driver as the argument
   c. Include super(driver); to refer to the parent class

```
WebDriver driver;

public BaseActions(WebDriver driver) {
	super(driver);
	this.driver = driver;

}
```

Step 11: Page Object Modelling and Page Factory

1. Create a package called cucumber.pages in main/java
2. Create 1 POM class for every stepDefenition and it should inherit BaseActions class
3. Create a global local variable for driver and create a constructor as below

```
public class CommonPage extends BaseActions {

WebDriver driver;

public CommonPage(WebDriver driver) {
	super(driver);
	this.driver = driver;
	PageFactory.initElements(driver, this);
}
```

4. Create PageFactory for all webElements and use By for locators
5. Below defined in the constructor to invoke the pageFactory
6. `PageFactory.initElements(driver, this);`
7. Use @FindBy annotation to store WebElements
8. Create a local driver object and use a constructor in Base Actions with `this.driver=driver;`
9. You will have to add `super(driver);` in the POM class constructor for BaseActions to use the same driver

Step 12 Hooks:

1. Create a class called 'Hooks' in cucumber.stepDefenitions
2. Hooks is a cucumber based Java class for Before and After methods
3. @After can be used for a method to close the driver and to reset the soft errors
4. @AfterStep can be used to hold a method which takes screenshot

```
@After
public void teardown()
{
	global.driver.quit();
}


@AfterStep
public void takeScreenshot(Scenario scenario) throws IOException
{
	Boolean takeScreenshot=global.takeScreenshot;
```

```
             WebDriver driver= global.driver;
             if (scenario.isFailed())
             {
                    File
sourcePath=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
                    byte[] filecontent=FileUtils.readFileToByteArray(sourcePath);
                    scenario.attach(filecontent, "image/png", "Screenshot
attached");
             }
             else if (takeScreenshot)
             {
                    File
sourcePath=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
                    byte[] filecontent=FileUtils.readFileToByteArray(sourcePath);
                    scenario.attach(filecontent, "image/png", "Screenshot
attached");

             }

      }
```

Step 13: Add Extent Reporting

1. Add a Extent.Properties file in src/test /resources with the below details
   - `basefolder.name=ExtentReports`
   - `basefolder.datetimepattern=d-MMM-YY HH-mm-ss`
   - `extent.reporter.spark.start=true`
   - `extent.reporter.spark.out=Reports/TestExecutionReport.html`
   - `screenshot.dir=Reports/screenshots`
   - `screenshot.rel.path=./screenshots/`
2. Add the following in cucumberOptions tag in runner file

```
@CucumberOptions(features = "src/test/java/cucumber/features",
glue="cucumber.stepDefinitions", tags= "@Regression", monochrome = true, dryRun =
false, plugin =
{"html:target/cucmberReport.html","json:target/cucmberReport.json",
      "com.aventstack.extentreports.cucumber.adapter.ExtentCucumberAdapter:"})
```

Additional: Parallel Execution

In runner class, add the below method to support parallel execution

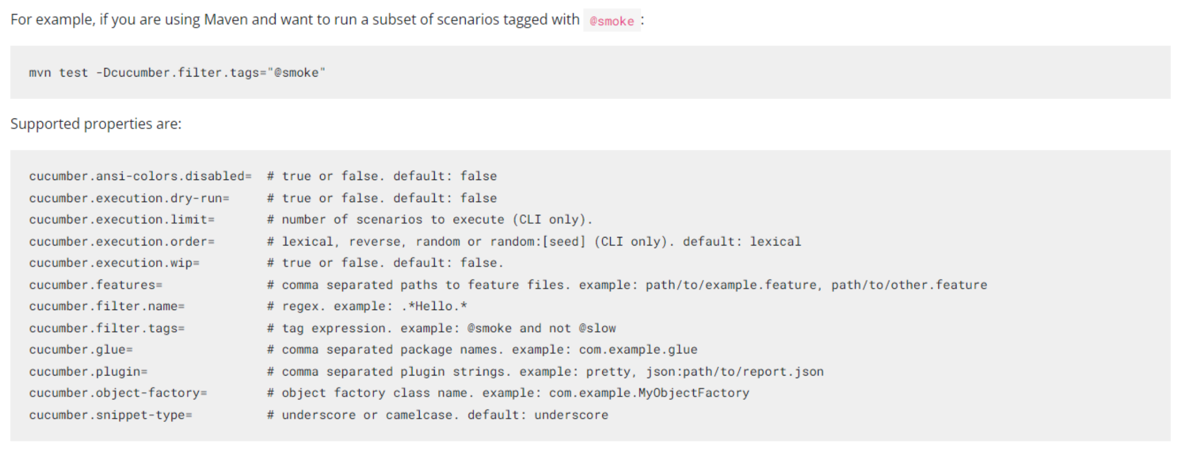
```
      /*
       * @Override
       *
       * @DataProvider(parallel = true)
       * public Object[][] scenarios() {
       * return super.scenarios();
       *
       * }
       */
}
```

Step 14: Running through Maven

1. Include this plugin in POM.XML to mark build as success even when there are test failures

```
<build>
  <plugins>
   <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-surefire-plugin</artifactId>
     <configuration>
       <testFailureIgnore>true</testFailureIgnore>
     </configuration>
   </plugin>
  </plugins>
 </build>
```

2. To run the framework using maven right click on the repository and click Run as-> Maven Build..
3. For a simple run we can give the goal as clean test
4. To run specific tags use test -Dcucumber.filter.tags="@Homepage"
   test -Dcucumber.filter.tags="@Homepage" -Dbrowser="chrome" -Denvironment="prod"
5. To run through command line-> open cmd-> cd to the directory where the framework resides and give mvn test -Dcucumber.filter.tags="@Homepage"



For example, if you are using Maven and want to run a subset of scenarios tagged with `@smoke`:

```
mvn test -Dcucumber.filter.tags="@smoke"
```

Supported properties are:

```
cucumber.ansi-colors.disabled=  # true or false. default: false
cucumber.execution.dry-run=     # true or false. default: false
cucumber.execution.limit=       # number of scenarios to execute (CLI only).
cucumber.execution.order=       # lexical, reverse, random or random:[seed] (CLI only). default: lexical
cucumber.execution.wip=         # true or false. default: false.
cucumber.features=              # comma separated paths to feature files. example: path/to/example.feature, path/to/other.feature
cucumber.filter.name=           # regex. example: .*Hello.*
cucumber.filter.tags=           # tag expression. example: @smoke and not @slow
cucumber.glue=                  # comma separated package names. example: com.example.glue
cucumber.plugin=                # comma separated plugin strings. example: pretty, json:path/to/report.json
cucumber.object-factory=        # object factory class name. example: com.example.MyObjectFactory
cucumber.snippet-type=          # underscore or camelcase. default: underscore
```

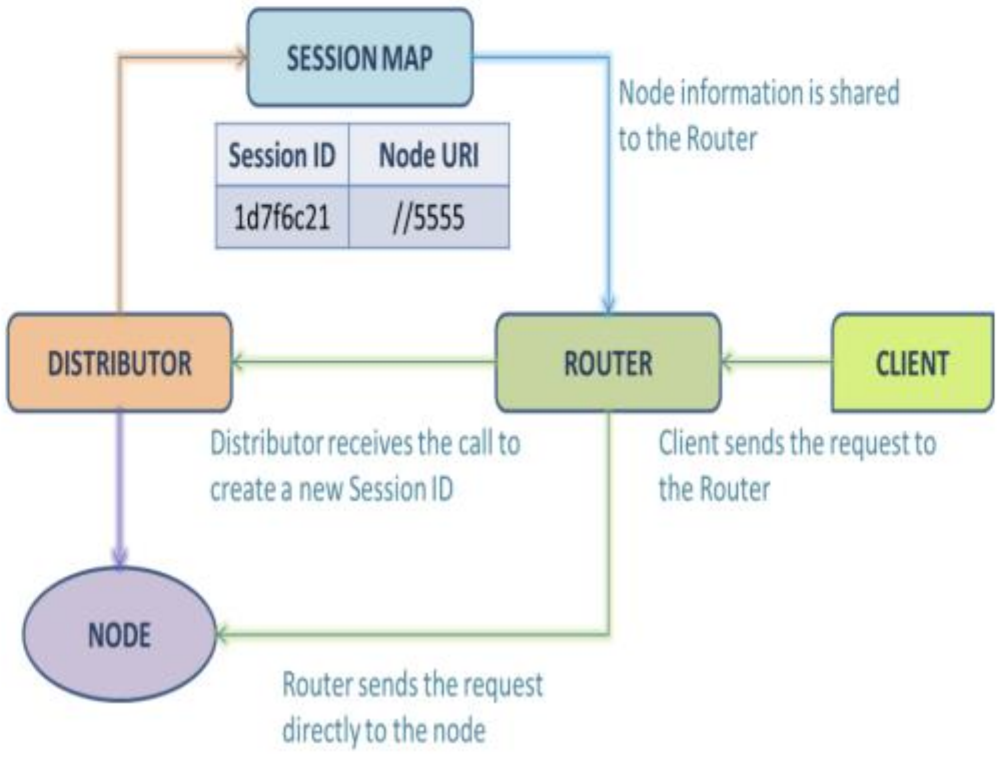
Refer to this URL: Cucumber Reference - Cucumber Documentation

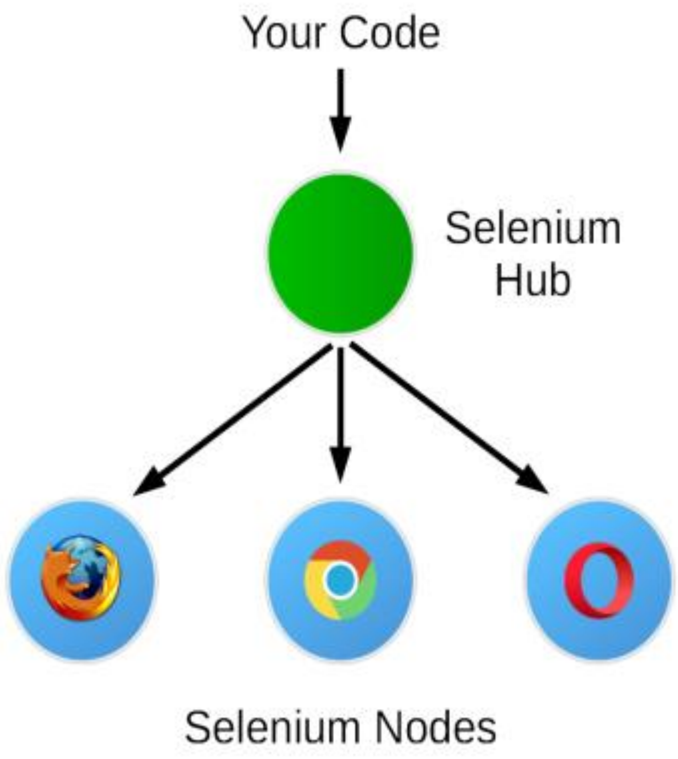
Note: Use DBrowser="browser_name" for maven control

Selenium Grid

1. Selenium Grid uses hub and nodes to start and distribute the running of selenium scrips

2. Selenium Grid is a smart proxy server that makes it easy to run tests in parallel on multiple machines


SESSION MAP
Session ID
Node URI
1d7f6c21
//5555
Node information is shared to the Router
DISTRIBUTOR
ROUTER
CLIENT
Distributor receives the call to create a new Session ID
Client sends the request to the Router
NODE
Router sends the request directly to the node



Steps to start selenium grid:

1. Download the Selenium Server jar and browser drivers and place it in the same folder in the Hub machine
2. Download the Selenium Server jar and browser drivers and place it in the same folder in the Node machine machine
3. Start the Hub - which eventually Starts Router, Distributor, Session Map , New Session Queue, Event Bus
   a. java -jar selenium-server-4.6.0.jar hub
4. To start the Node in Same Machine where Hub is running
   a. java -jar selenium-server-4.6.0.jar node --detect-drivers true
5. To start the Node in different Physical Machine

   java -jar selenium-server-4.6.0.jar node --detect-drivers true
   -- publish-events tcp://<XPUB_address> --subscribe-events tcp://<XSUB_address>

   Note: Check the Status of Grid with http://localhost:4444/ (default port)

Framework Level changes:

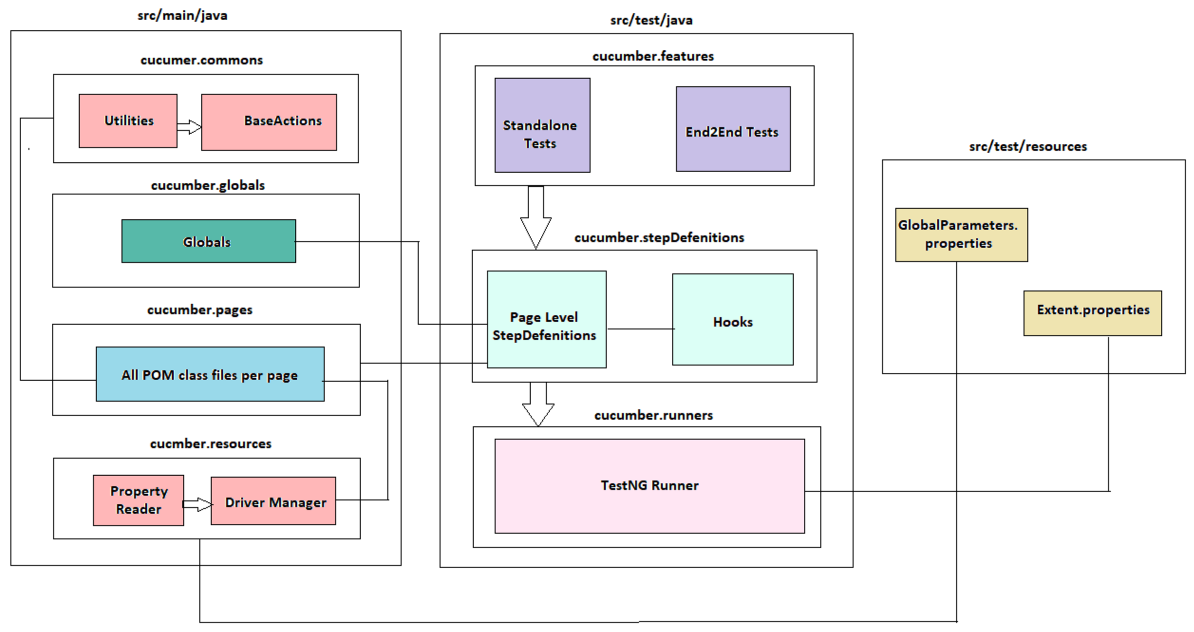
1. Add selenium server dependency in maven POM.XML
2. Add below parameters in GlobalParameters.properties

```
#GRID PARAMETERS
## Options-true, false
GRID_Execution=true
GRID_HUB=http://192.168.29.232:4444
```

3. If GRID_Execution is set to true below code should be invoked in invokeBrowser() method in DriverManager class

```
if(gridExecution.equalsIgnoreCase("false"))
        {
        }
else
        {
                String hub= readPropertyValue("global", "GRID_HUB");
                DesiredCapabilities capability= new DesiredCapabilities();
                capability.setBrowserName(browser);
                driver= new RemoteWebDriver(new URL(hub),capability);
        }
```

Final Architecture:


src/main/java
cucumer.commons
Utilities
BaseActions
cucumber.globals
Globals
cucumber.pages
All POM class files per page
cucmber.resources
Property Reader
Driver Manager
src/test/java
cucumber.features
Standalone Tests
End2End Tests
cucumber.stepDefenitions
Page Level StepDefenitions
Hooks
cucumber.runners
TestNG Runner
src/test/resources
GlobalParameters. properties
Extent.properties